Dynamic update of a virtual cell for programming and safe monitoring of an industrial robot

A. Ferraro * **M. Indri** * **I. Lazzero** *

* *Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy*
e-mail: andrea.ferraro.rrg@gmail.com, {marina.indri, ivan.lazzero}@polito.it

Abstract: A hardware/software architecture for robot motion planning and on-line safe monitoring has been developed with the objective to assure high flexibility in production control, safety for workers and machinery, with user-friendly interface. The architecture, developed using Microsoft Robotics Developers Studio and implemented for a six-dof COMAU NS 12 robot, established a bidirectional communication between the robot controller and a virtual replica of the real robotic cell. The working space of the real robot can then be easily limited for safety reasons by inserting virtual objects (or sensors) in such a virtual environment. This paper investigates the possibility to achieve an automatic, dynamic update of the virtual cell by using a low cost depth sensor (i.e., a commercial Microsoft Kinect) to detect the presence of completely unknown objects, moving inside the real cell. The experimental tests show that the developed architecture is able to recognize variously shaped mobile objects inside the monitored area and let the robot stop before colliding with them, if the objects are not too small.

Keywords: Virtual robotic cell, robot monitoring, depth sensors

1. INTRODUCTION

In the present market situation many small and medium-sized manufactures focus their efforts on providing their costumers with highly specialized products, suited for specific demands. Consequently the production volume of a single piece tends to remain small, due to frequent changes of materials and processing cycles.

The use of industrial manipulators is currently subordinate to conditions not compatible with such environments. To introduce automatic machining in the work flow, the interested part of the manufacturing chain must be enclosed in locked cells, equipped with safeties that cause a production stop if the cell is opened. Another relevant issue is the necessity of accurate and time-consuming re-configurations required to perform even small changes in the work cycle, which in a competitive environment can quickly become burdensome.

In this situation SMEs are looking for robotic solutions that allow to share spaces among different manufacturing cycles and possibly human workers, able to switch to new productions without needing long reconfigurations and bundled with easy to use interfaces.

Several commercial robot programming frameworks are available, such as SimStation for graphical programming or ROBCAD, a workcell simulation tool by Tecnomatix Technologies Ltd., but they allow only the off-line motion planning and the application of the planned trajectory to the real robot, without the possibility to use their virtual environment to monitor the actual robot motion, since communication is established in one direction only, from the virtual environment to the manufacturing cell.

Currently Politecnico di Torino in collaboration with Comau S.p.A. (`http://www.comau.com`) is working on a hardware/software architecture, developed using Microsoft Robotics Developers Studio (MRDS) to program and safely handle a manipulator motion in open spaces, aimed at guaranteeing easy and quick programming, and expanded possibilities in a scenario in which space is shared among different robotic systems, with possible interaction with human operators.

The developed solution consists in a virtual cell in MRDS environment, which unites the functionalities of a standard off-line simulator with soft realtime robot monitoring and the possibility of controlling the real manipulator, thanks to the bidirectional communication established between the robot and the virtual environment.

This peculiar setup allows for the switching master-follower control routine called *Viceversa* to take place: the simulator supports the creation of virtual obstacles which, as in most traditional controllers, can send stop signals when contacted by parts of the virtual robot, replicating as follower the motion of the real one. After the contact, the virtual system becomes the master and plans a path to a safe position, which is then tracked by the real robot, now become the follower.

The current experimental setup includes as real robot a standard Comau Smart NS 12-1.85 anthropomorphic arm, coupled with a C4G industrial controller, whose external communication link can be used to read status information, exchange data and give instructions to the robot.

The results achieved so far were published in papers Abrate et al. (2010) and Abrate et al. (2011). The first one describes how it is possible to create the virtual cell within MRDS environment, with a physics engine able to handle all the variables needed for a realistic replica of a work environment. The resulting simulator can be used as off-line programmer in joint space and as an external guard, monitoring or directing the robot's

movements through a soft-realtime connection, as desired. Programming in the Cartesian space was subsequently added, thanks to the implementation of the efficient, iterative inverse kinematics procedure, described in Abrate et al. (2011).

Tests conducted on the virtual cell underlined the effects of the system structural delays, due mainly to the controller architecture, and to the technical time required by the robot to go from motion to complete stop (see Abrate et al. (2011) for a complete description of the robot stopping routine). This situation often prevented the real robot from stopping before colliding with the obstacle. To address such issue, "risk layers" have been introduced in Abrate et al. (2011), consisting in concentric, navigable boxes wrapped around the virtual obstacles. The consequence of risk layers crossing is a progressive slowdown of the real arm motion, inversely proportional to the distance between it and the obstacle, thus reducing the delay effects to a safe level.

This paper focuses on the usage of low cost depth sensors to make the MRDS environment aware of the presence of obstacles inside the real working area and consequently automatically update the virtual cell status.

So far every object inside the virtual cell had to be explicitly introduced and fully described by the user; such a procedure, while adequate for a static environment, cannot handle quickly and easily the insertion of new elements in the working area. Furthermore, a real-time monitoring of the environment makes possible to track automatically mobile objects inside the working area.

Many research groups worked on robot surveillance systems involving multiple standard cameras, such as Bosch and Klet (2010), Ebert and Heinrich (2001) and Krüger et al. (2004); these methods proved to be reliable in the correct situations, but all of them require a mandatory step of image processing. The usage of a depth sensor allows to keep a simpler prototypal setup and the direct availability of a three-dimensional view, lowering the computational demand, makes possible to run all the virtual cell functions on a single standard computer. The usage of depth sensors to create real time virtual replicas of robotic cells has been explored in Graf et al. (2010), but the work is more focused on the detailed characterization of the detected obstacles than on the creation of constraints to the manipulator motion.

For the experimental setup a commercial Microsoft Kinect depth-sensor has been chosen, due to its particular features of low cost and wide availability. The pre-existing structure of the virtual cell has been implemented using MRDS Distributed Software Services (DSS), which allows to create distinct but communicating services to handle the various functions of the system. The most recent MRDS release gives native support to Kinect integration in such environments.

The paper is organized as follows: Section 2 illustrates the structure of the prototype and the preliminary tasks performed; Section 3 describes the processing of the depth camera data and the handling of the virtual objects; Section 4 reports the results of some experimental tests, aimed at assessing the system's fuctionality. Section 5 draws some conclusions and illustrates the guidelines of current and future works.

2. WORKING AREA SETUP AND CALIBRATION

To introduce the depth sensor avoiding compatibility issues, the virtual environment has been migrated to Microsoft Robotics Developer Studio R4 Beta, which offers native support for the Kinect depth camera. The distance measurements are provided by an IR-projector/IR-sensor couple, both mounted on the depth camera body. Such setup could easily create interferences when multiple cameras are placed in the same environment, since it would be possible for an IR-sensor to pick projections from the other camera instead of it's own reflections. To avoid any possible self-induced noise in the prototype, exploit the biggest possible vision field and reduce the amount of data to be processed at every iteration, we have chosen to place a single sensor in the center of the robotic cell roof, as depicted in Fig 1.

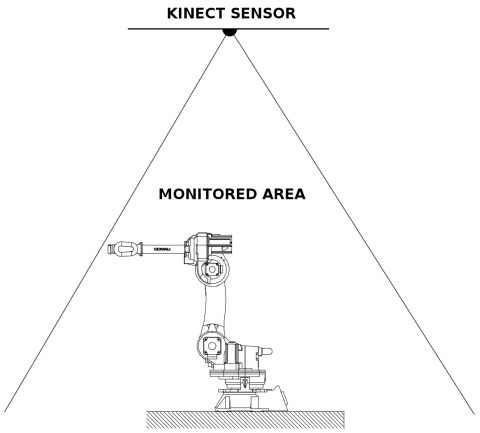


Fig. 1. Prototype depth camera placement

A single point of view in a different position would have created difficulties in evaluating the effective depth of the detected obstacle, being available only distance data related to one side of the object. A vertical placement instead allows to approximate the obstacles with columns with a minimal reduction of the robot operating area. The implemented solution used to employ an estimation method based on the depth value of the lowest perceived point of the object. Such a solution resulted to be acceptably accurate in several cases, but showed some problems with irregular objects, for which the obtained values could be not fully reliable, due to the lack of information about the structure of the inferior part of the obstacle, like in the situation sketched in Fig 2. In Section 3 the procedure will be described and applied to correct the image perspective error.

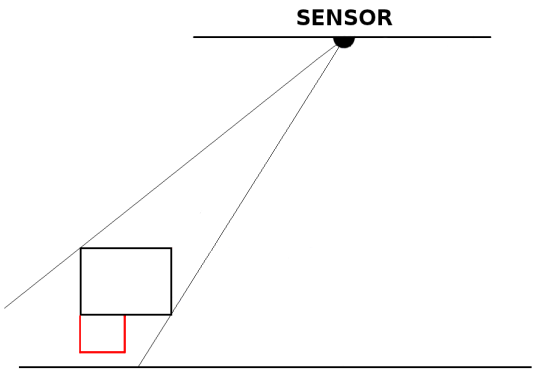


Fig. 2. Object with irregular underside, the section in red cannot be seen by the depth sensor

The Kinect camera performs autonomously the conversion between raw data collected by the distance sensor and a "more" usable distance from the plan on which it lies (Fig. 3). Given that, only a conversion ratio between depth values and meters is required to perform the correct conversions between depth data and actual distances in the world. Such a calibration step is always necessary for a new Kinect, since every camera is slightly different.

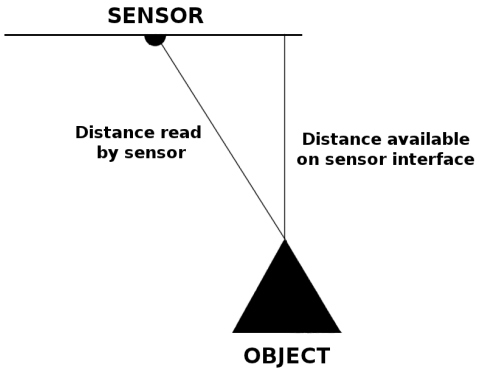


Fig. 3. Autonomous processing of distance data by the Kinect depth camera implementation

The calibration of the prototype has been performed fixing a panel of known size on the operating end of the monitored robot, and using it to collect samples of its perceived size at various distances from the sensor. Using the robot as a support ensured a high degree of repeatability.

The full calibration produced eight samples for each dimension (width and height of the object and distance from the sensor) collected starting with the robot in full vertical pose and lowering the operating end at steps of 10 cm each until reaching the joints stroke end. With those values it has been possible to obtain the fitting first grade polynomials (Fig. 4) to estimate the conversion ratio between depth values and real world distances, and between objects sizes in depth frames and real world objects sizes. The coefficients of such fitting functions have then been used in the following equations to obtain the real world dimensions of a depth map object:

$$\mathbf{X} = \mathbf{x} * (\mathbf{depth} * \mathbf{xmul} + \mathbf{xadd}) \quad (1)$$

$$\mathbf{Y} = \mathbf{y} * (\mathbf{depth} * \mathbf{ymul} + \mathbf{yadd}) \quad (2)$$

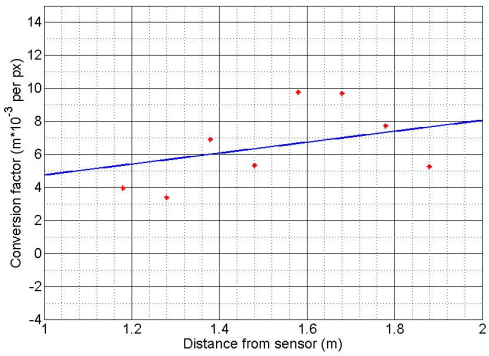
$$\mathbf{Z} = \mathbf{depth} * (\mathbf{depth} * \mathbf{zmul} + \mathbf{zadd}) \quad (3)$$

where **X** and **Y** are the dimensions (width and length) of the object, **Z** its distance from the sensor in the real world, **x**, **y** and **depth** are respectively width, height and distance from the sensor of the object in the depth map, **xmul**, **ymul** and **zmul** are slopes of the fitting lines and **xadd**, **yadd** and **zadd** their offsets.

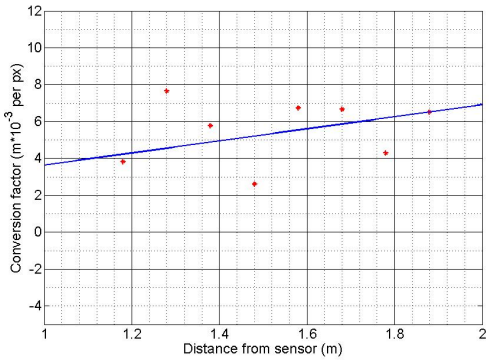
3. OBJECTS IDENTIFICATION AND HANDLING

The Kinect camera provides depth data in the form of an array of 16-bit unsigned integer values, which are used in the virtual environment in two separate ways:

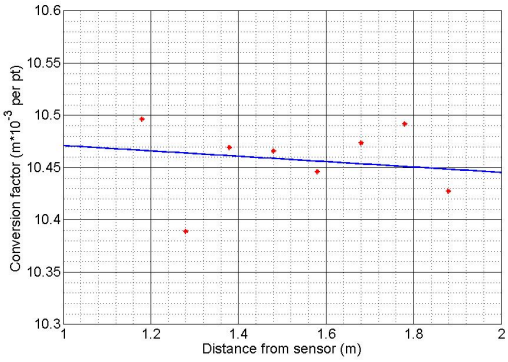
- In the same data type, after reshaping in matrix form for ease of reference, as base values for conversions between depth data and world distances
- Converted to bitmap to create a frame adequate for graphic processing



(a) Graph of the conversion factor for the X axis of the Kinect reference system



(b) Graph of the conversion factor for the Y axis of the Kinect reference system



(c) Graph of the conversion factor for the distance from sensor of the Kinect reference system

Fig. 4. Fitting polynomials for the conversion rates between depth values and real world dimensions

A graphical processing phase is necessary after the conversion of the depth data into a grayscale depth image. The open library AForge.NET (`http://www.aforgenet.com`) has been used, due to the availability of a C# native implementation and to the presence of optimized filters matching the required tasks. Other libraries and algorhitms oriented to a more detailed analysis of the detected objects (like the procedures introduced in Graf et al. (2010)) have not been employed to keep low processing times and to allow a faster prototyping.

The Kinect sensor perception field ranges from 0.8 m of distance from the IR projector to about 4 m; every point out of it is set to 0 in the depth map. In order to obtain cleaner object detection a *floor* variable that can be set

by the user has been introduced, to represent a bottom perception level which, substantially, shortens the Kinect field of view to the given value. A secondary benefit is the possibility to ignore all the objects under a given height, at user discretion. This could be useful to avoid the creation of undesired virtual obstacles (with the corresponding risk layers) for small, negligible entities, such as robot wiring and floor irregularities.

The following steps are performed on every processed frame:

- Points without a readable value due to IR reflections are set to ground level.
- Every point equal or farther than the configured level of *floor* is set to black.
- The graphic entity matching the robot position is recognized and graphically filtered out of the image.
- The remaining graphic entities (blobs) are recognized and listed in a dedicated data structure of the MRDS environment.

The resulting data are used as source to create the actual objects to be added to the virtual cell, after a further non-graphic filtering to rule out blobs too small to be considered other than noise. For each blob, the distance from the Kinect and the size on the image are processed according to (1) - (3) to obtain the characterizing values of size and position of the virtual object. Before their inclusion into the virtual environment, a dimensional corrective step is performed to avoid widening effects due to sensor's perspective, as it can be seen in Fig. 5. The sizing error is calculated geometrically according to the scheme in Fig. 6 using the following equations:

$$\beta = acos\left(\frac{\mathbf{X_1}}{\sqrt{\mathbf{X_1^2} + \mathbf{Z_1^2}}}\right) \tag{4}$$

$$\mathbf{X_2} = \frac{(\mathbf{Z_2} - \mathbf{Z_1}) * cos\beta}{sin\beta} \tag{5}$$

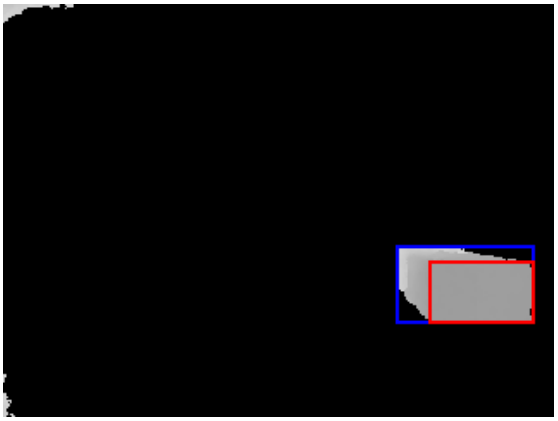


Fig. 5. Filtered depth image: lined in red the actual object's size, lined in blue the oject's size as registered by the graphic library

In this case the possibility of hidden sections (Fig. 2) is no longer an issue, since all the calculation is aimed at correcting the size of the object as perceived by the sensor itself.

3.1 Detection of moving objects

The procedure described so far provides satisfactory results in case of static objects. Some further difficulties arise when moving objects are to be correctly detected and

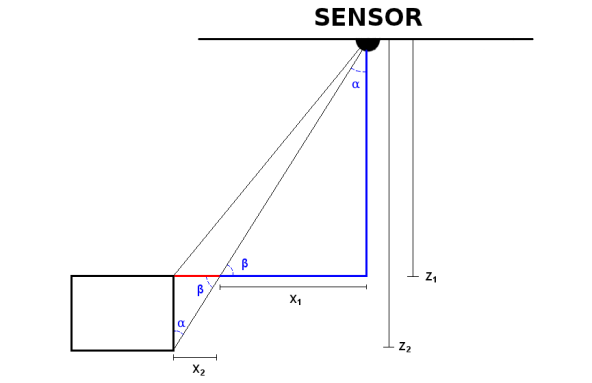


Fig. 6. Size correction scheme: X_1 is the graphical object's distance from the frame center, X_2 is the size error, Z_1 is the distance from the sensor of the object's top point and Z_2 is the distance from the sensor of the object's lowest point seen by the sensor

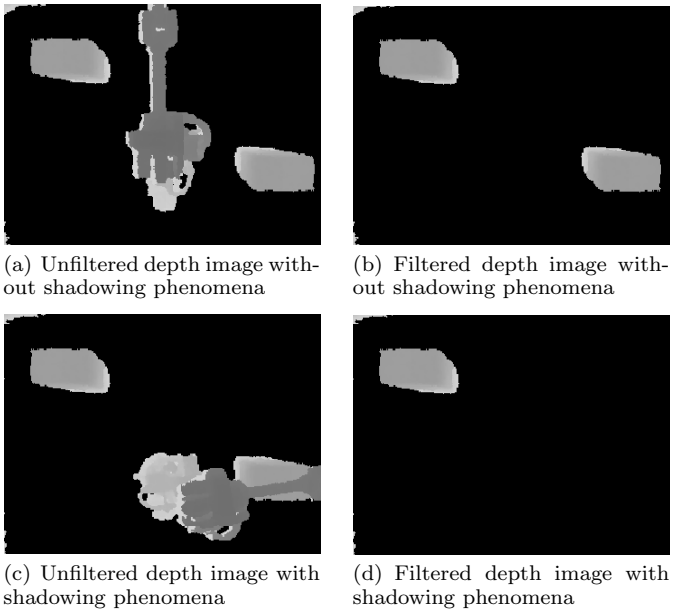
inserted into the virtual cell. In particular in the search for the lowest point of *floating* objects, a varying number of border points is often marked with depth values between the expected ones and the underlying floor level. A possible criterion, to prevent these points from influencing the results of the sizing error computation (4) and (5), is based on the heuristic definition of the maximum difference between depth values of adjacent points belonging to the same object. To ensure the exclusion of misread values the threshold D, representing such a maximum difference, is used in two separate validation steps:

Peer check: the difference between the point depth value and the depth values of the surrounding object points must be lower than D.

Object check: the difference between the point depth value and the depth value of the point of the object closer to the depth sensor must be lower than D multiplied by the distance between them.

The value of D can be heuristically estimated on the basis of the maximum observed difference between the values of two adjacent points of a vertically sided object in the depth bitmap, plus a safety margin. Underestimating the value would lead to the exclusion of correct points, and consequently to the introduction in (5) of reduced values of Z_2; an overestimate of D would be less critical, since the misread values tend to be significantly closer to the underlying level than to the object border.

The motion of the object revealed to be not easily manageable on the side of the virtual environment. Moving and resizing existing objects is not possible since it causes a partial loss of the physical properties of the risk layer entities, so it is necessary to delete and recreate an object every time it moves or changes its shape. The complete deletion of pre-existing blobs and consequent creation of new ones for every processed frame would represent a very easy approach, requiring no computation to discriminate mobile objects from still ones. Unfortunately the refreshing of every visible object for each frame resulted to be too burdensome for the MRDS simulator. A feasible solution is instead based on the implementation of proper functions to discriminate between obstacles which need to be refreshed and objects remained still since the last frame. This solution allows to maintain the simulator responsive through the refresh cycles and to avoid any lag in handling the more critical collision messages.



(a) Unfiltered depth image without shadowing phenomena

(b) Filtered depth image without shadowing phenomena

(c) Unfiltered depth image with shadowing phenomena

(d) Filtered depth image with shadowing phenomena

Fig. 7. An example of the possible effects of shadowing: in (c) the obstacle is considered by the graphic library part of the robot, and consequently removed from the image (d)

The use of dedicated data structures to keep track of objects positions and movements allows also to introduce some simple rules for the exit of an obstacle from the monitored area. Preventing a virtual object to be deleted unless close to the area border, or moved to a nearby location, is instrumental to avoid troubles from possible signal glitches. It also forbids virtual obstacles to disappear when their real counterpart enters in the shadow created by a larger object. Such an occurrence is particularly troublesome when an obstacle is shadowed by the moving robot itself, becoming, from the graphic library point of view, part of the same blob and consequently an element to be removed from the image toghether with it (Fig 7).

4. EXPERIMENTAL RESULTS

The final prototype has been tested experimentally with obstacles in the real robotic cell, moving both vertically and horizontally (Fig. 9).

The video "Self updating virtual cell collision tests with moving and floating obstacles", containing samples of the testing phase, is available in the "Video" page of the website `http://www.polito.it/labrob`, demonstrating the dynamic behaviour of the prototype in various conditions of motion speed and obstacle positioning. In the first part of the video is clearly visible the slowdown of the manipulator as its distance from the moving obstacle decreases, while the second part depicts two tests, respectively at 20% and 40% of the maximum speed of the robot (2 rad/s), involving a smaller floating obstacle. Both sections include an initial part, which shows how the virtual cell updates automatically position and dimensions of the moving obstacles.

The video shows that the developed system is able to detect the presence of the objects, place them correctly in the virtual space, and assign them the correct dimensions.

The static collision tests performed at velocities greater than 30% of the maximum robot speed showed the inadequacy of uniformly spaced risk layers (like in Fig. 10(a)) in such conditions. To completely avoid the risk of contact

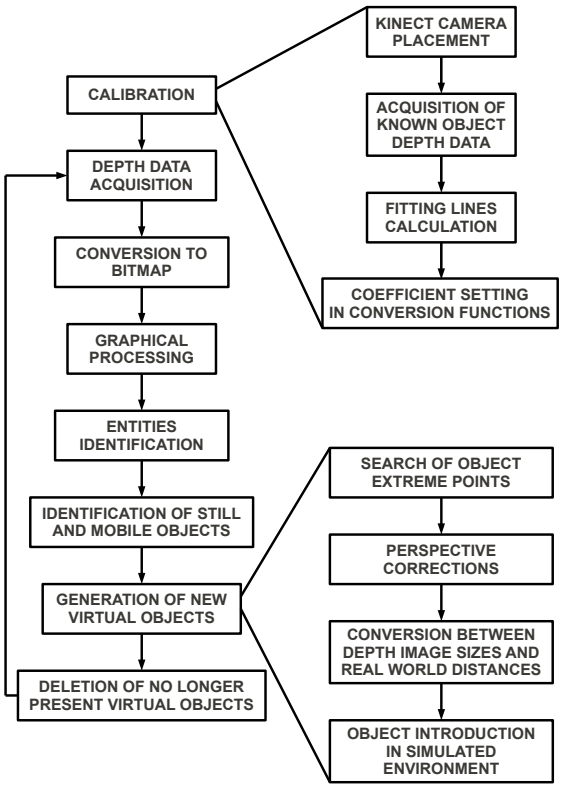
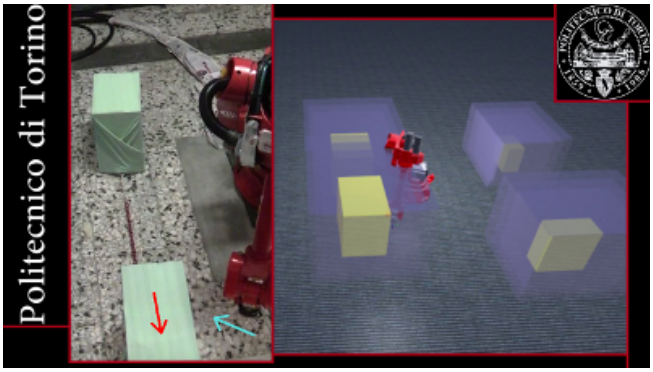


Fig. 8. Setup and elaboration cycle of the automatic update prototype

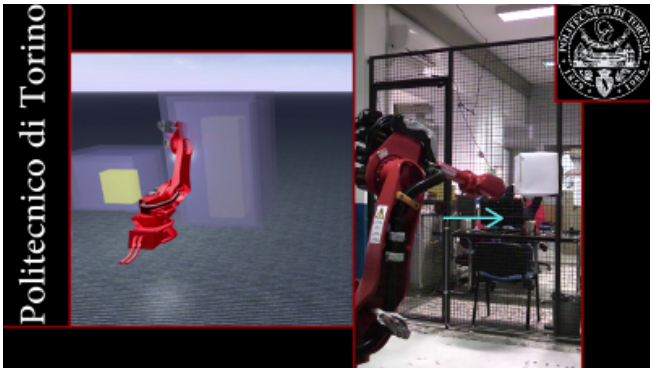
with the obstacle it was necessary to modify the original risk layers spacing (adopted in the tests reported in Abrate et al. (2011)), expanding the inner layers as in Fig. 10(b), in order to obtain a quicker slowdown: Figs. 10(c) and 10(d) compare the original and the modified slowdown profiles corresponding to the inserted risk layers. The innermost yellow layer (corresponding to the complete stop of the robot) must be placed at least at 5 cm of distance from the actual object surface, according to the maximum stopping time imposed by law for industrial manipulators.

With this configuration the prototype had been able to successfully stop without coming in contact with the object, regardless of the approach vector. A test with obstacles moving at speeds up to 0.5 m/s did not reveal significant differences in the stopping times, but evidenced issues caused by the shadowed areas of the image. As a matter of fact every big object in the working area can hide a smaller one from the sensor, but the problem becomes evident when the shadow is generated by the robot itself which, being a medium-sized anthropomorphic arm, can hide small objects while approaching them. The solution described in Section 3.1 allows to keep track of the object's last known position and size, preventing the robot to enter their areas also when real time data about it are no longer available from the depth sensor.

During the tests has also become evident that surfaces which do not provide a good IR response (black painted ones in general, expecially if polished) should be avoided whenever possible, since a lack of reflection from a point puts it automatically beyond the maximum readable distance for the Kinect camera, and such a discontinuity is not easily handled in the virtual object's generating code.



(a) Robot approaching horizontally moving obstacle



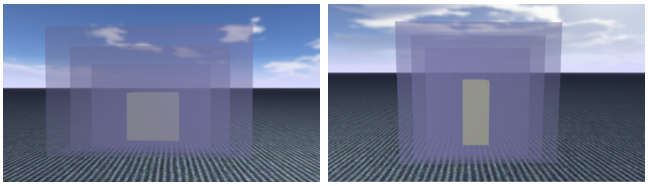
(b) Robot approaching floating obstacle

Fig. 9. Synchronized views of the real and virtual manipulators approaching a ground moving object (a) and a floating one (b), motion directions are shown by the red (obstacle) and cyan (robot) arrows

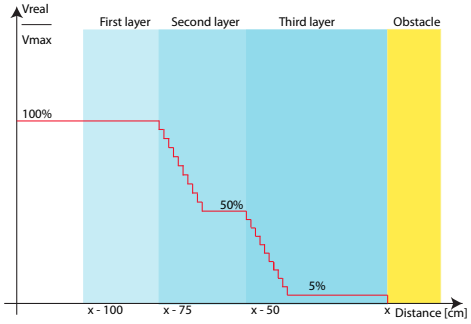
5. CONCLUSIONS AND FUTURE WORKS

The experimental test showed that the developed architecture is able to recognize variously shaped mobile objects inside the monitored area and let the robot stop before colliding with them, if the objects are not too small. The current setup, with a single low-cost depth sensor devoted to the detection of completely unknown objects, is not sufficient to ensure complete safety conditions in any situation, but provides a tool to easily and dynamically reduce the operating area of the robot, so to avoid undesired contacts with objects in the cell. Possible enhancements of the current architecture towards safe conditions include the usage of predictive algorithms to estimate the future position of the detected obstacles, and the coupling of the depth map with data collected by other kinds of sensors differently placed in order to eliminate the shadowing phenomena previously described. Refinements of the image processing could produce more accurate data about shape and position of the real obstacle, allowing for smaller bounding boxes, and therefore a better definition of the safe operating area of the robot.

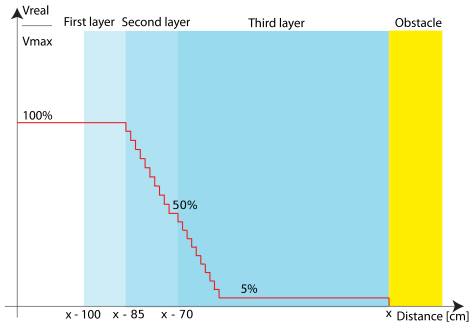
Future works will be devoted also to the definition of proper re-planning procedures including obstacle avoidance algorithms, after the detection of a possible contact with an object, in order to continue as much as possible the pre-assigned working cycle. In the current implementation, the robot simply stops after the slowdown behavior imposed by the risk layers crossing, goes back at a reduced velocity to a "safe" target along the previously tracked trajectory, and waits there for further instructions.



(a) Original risk layers spacing (b) Modified risk layers spacing



(c) Original slowdown graph



(d) Modified slowdown graph

Fig. 10. The modified risk layers spacing causes a quicker slowdown of the arm motion, granting full stop before coming in contact with the obstacle also for tests over 30% of the manipulator maximum speed

REFERENCES

Abrate, F., Bona, B., Indri, M., Messa, D., and Bottero, A. (2010). Motion planning and monitoring of an industrial robot via a virtual manufacturing cell. In *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, 67–72.

Abrate, F., Indri, M., Lazzero, I., and Bottero, A. (2011). Efficient solutions for programming and safe monitoring of an industrial robot via a virtual cell. In *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2011)*, 434–439.

Bosch, J. and Klet, F. (2010). Safe and flexible human-robot cooperation in industrial applications. In *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 107–110.

Ebert, D. and Heinrich, D. (2001). Safe human-robot-cooperation: problem analysis, system concept and fast sensor fusion. In *IEEE Con. Multisensor Fusion ans Integration for Intelligent Systems*, 107–110.

Graf, J., Dittrich, F., and Wörn, H. (2010). High performance optical flow serves Bayesian filtering for safe human-robot cooperation. In *ISR / ROBOTIK 2010*, 1–8.

Krüger, J., Nickolay, B., and Schulz, O. (2004). Image-based 3D-surveillance in man-robot-cooperation. In *INDIN '04 2nd IEEE International Conference on Industrial Informatics*, 411–420.